Final Report

Nov 18, 1999

Sponsored by

Defense Advanced Research Projects Agency
ITO (Col. Mark Swinson)

ARPA Order No. D611, Amdt. 56

Issued by U.S. Army Aviation and Missile Command Under

Contract No. DAAH01–99–C–R140

Name of Contractor: *Clifton Labs, Inc.*
Business Address: 3678 Fawnrun Dr.
Cincinnati, OH 45241

Principal Investigator: Dale E. Martin
Phone Number: (513) 563-4731

Effective Date of Contract: 12 April 1999
Contract Expiration Date: 18 Nov 1999

Short Title of Work: **Elastic A Framework for Self-Adaptive Software**
Reporting Period: 12 April 1999–18 Nov 1999.

The Contractor, Clifton Labs, Inc, hearby declares that, to the best of its knowledge and belief, the technical data delivered herewith under Contract No. DAAH01–99–C–R140 is complete, accurate, and complies with all requirements of the contract.

Date: November 18, 1999

Name and Title of Authorized Official: _____
Philip, A. Wilsey, President

Approved for public release; distribution unlimited.

19991126 060

Abstract

The Elastic program is a SBIR Phase I effort to develop and deploy a toolkit for constructing self-adaptive software systems. The purpose in building this toolkit is to enable the automatic optimization of parallel software systems. Insertion of self-adapting technology into such systems will allow them to react to changes in the execution environment in order to sustain and increase performance.

The Elastic Toolkit will allow the developer to control their software in a variety of ways. First, parameters known to affect system performance can be monitored through the insertion of software sensors; as these parameters are monitored, control actions can take place to affect performance. An example of this type of control would be the automatic adjustment of communication parameters in a message passing program. Another example of the type of control that Elastic will allow is the automatic selection of algorithms and data structures, based on parameters specified by the user.

The Elastic approach differs from traditional off-line postprocessing techniques in that it occurs automatically at runtime. This has the advantage that an application will realize performance benefits immediately upon being moved to a new hardware/software platform; traditional platform-specific "tweaking" becomes less important.

Initial data collected as part of the Elastic Phase I effort showed that a synthetic application sensitive to locking can derive as much as a 13X speedup simply by choosing a different locking algorithm. While results on a real application probably will not be as dramatic as those of the synthetic application, it shows that there is potential for performance gains from simple algorithmic optimizations.

1 Introduction

Clifton Labs is investigating methods for evaluating the performance of adaptive software. Automatic methods for the construction of performance sensors and evaluators are being investigated. Investigation of predictive and recovery based control policies will also be performed. Leveraging this research, a model configuration for a toolkit for constructing self-adaptive software systems will be developed. Support for building new control techniques will also be provided. The toolkit will then be applied to realistic large scale problems to demonstrate the applicability of the framework. Iterative refinement of the on-line configuration control system will be performed to improve its applicability to several problem domains. Finally, leveraging the experience gained from the construction and fielding of the toolkit, performance metrics and success criteria for self-adaptive software will be developed.

2 Major Accomplishments

The goal of this Phase I effort, as with all such efforts, has been to assess the feasibility of the proposed work. In addition, the commercial viability of such an effort has been assessed. Both goals have been achieved — the Elastic project is both technically feasible, and commercially viable. The details of our efforts can be found in the following sections.

Much of our early effort in assessing feasibility went into researching past projects applicable to the problem domain. Researching other's efforts was part of our strategy in order to assess feasibility, market, competition, and other important aspects of product development.

Another area of early focus was that of commercial viability. One of the manners in which we addressed this problem was to send an informal survey to leading companies in the computer and/or software industries. The results from this survey were overwhelmingly positive and will be presented in this report.

Lastly, prototypes of key pieces of the Elastic Toolkit have been designed, implemented, and tested. Going through the design and construction of these pieces has allowed us to understand many of the cardinal issues involved. Furthermore, having a working prototype has allowed us to perform testing that is not possible in any other manner.

Detailed accomplishments on a per task basis appear in the following sections, and a short summary follows:

- Commercial and Technical Feasibility: Extensive research of existing systems, informal surveying of major hardware and software vendors, and prototype design and implementation have all been done to establish commercial and technical feasibility. Results of all activities have been positive.
- Prototype Design and Implementation: Key portions of the toolkit have been designed and implemented to the point that they are useful for testing and demonstration of viability.
- Testing: Using the prototype toolkit, testing has been performed that confirms the usefulness and performance realization of Elastic technology.

As one small step in addressing commercial feasibility, Clifton Labs executed an informal survey among members of the technical staffs at a number of large companies and organizations involved in significant software engineering projects. The questions were aimed to assess the interest in Elastic technology and multithreaded programming in general. The results of this survey were overwhelmingly positive, with interest expressed from individuals at the following organizations:

- AT&T Bell Labs, NY
- AT&T, Cincinnati, OH
- Fore Systems, Inc
- NASA Ames Research Center, CA
- SDRC, Cincinnati, OH
- Viewlogic Inc., MA

Two responses of "not interested" were received. In both cases, the individuals were not involved in projects relevant to Elastic Technology, *i.e.*, they were working on hardware projects.

Another interesting question that has arisen about the business aspect of Elastic is whether or not selling programming libraries could be profitable. In other words, would there be any customers willing to purchase programming libraries?

In fact, several programming libraries have been successfully developed, marketed, and sold. One good example of a company selling several such products is Rogue Wave Software, which reports an earned revenue of $53.1 million for the fiscal year 1999.

Rogue Wave offers several C++ libraries that have been extremely well received by the programming community, including the ubiquitous Tools.h++ library. In addition to selling directly to the programming community, Rogue Wave libraries have been licensed to companies like Borland and Watcom for insertion into their integrated development environments. Thus, a market for programming libraries already exists and a full development and deployment of the Elastic optimization library would fit into this market.

3 Schedule Adherence

All tasks have been completed on schedule.

4 Significant Adjustments to Project Plan

Upon meeting with the new DARPA Program Manager, Dr. Janos Sztipanovits, adjustments have been made to the project plan. As requested, the focus of our research has been narrowed to a specific domain. The domain that we have chosen to focus on are components and data structures relevant to parallel processing; specifically we have implemented locks and concurrent list structures.

5 Current Status & Plans by Project Tasks

5.1 Task 1: Evaluation of software functionality and performance at run-time

Runtime evaluation of software functionality and performance is a key aspect to the Elastic tool suite. In order to perform adaptive control on a software system, an accurate assessment of its performance must first be made. Only then can control decisions be made.

Early effort focused on understanding what has been done by other groups, both in the commercial and in the research communities. Later effort included designing "Profiling Elements" which could be transparently inserted into an application and generate statistics of the usage patterns of specific instances of data structures. Locks, for instance, are evaluated for the amount of contention, average wait time of each thread, read/write ratios, and several other parameters.

Accomplishments:

- Evaluation of Existing Methodologies: Much time has been spent evaluating the strengths and weaknesses of existing parallel performance monitoring systems, both in the commercial and research arenas. Commercial systems studied include:
  - *DEEP*, by Pacific-Sierra Research.
  - *Insure++*, *TCA*, *INUSE*, by Parasoft.
  - *Clustor*, by Active Tools.
  - *VAMPIR*, *VAMPIRtrace*, and *DIMEMAS*, by Pallas.

  Research systems studied include:
  - *AutoPilot* and *SvPablo*, by University of Illinois' Pablo Research Group.
  - *BRISK*, by Michigan State University's Performance Gateway (PG^{RT}) Research Group.
  - *FALCON*, by Georgia Tech's Systems Research Group.
  - *Paradyn*, by University of Wisconsin's Paradyn Research Group.
- *Design of Profiling Elements*: Elements of the prototype Elastic Toolkit were designed using the "Bridge" design pattern, decoupling the interface from the implementation. This allows the insertion of any implementation of an element for a specific instance of that element. Taking advantage of this design, elements that could profile access patterns were designed such that they are completely transparent to the application being profiled.

- *Design of Portable High Resolution Timers*: Many interesting performance characteristics have to do with the measurement of time. Unfortunately, due to the wide variations of timing hardware available on computer systems, no standard mechanism for realizing high resolution timing is currently available. However, most systems have a non-portable mechanism for achieving such timings. As part of our prototype system, we have developed a portable, object oriented mechanism for achieving high resolution timing across several platforms. Furthermore, support for this mechanism should be easily portable to other platforms, and fallbacks to default mechanisms are in place.

The first profiling element developed was the `ProfilingLock`. The profiling lock collects information on a number of critical parameters that can be used to classify the context in which the lock is being used.

Lock Overhead: The amount of time required to acquire and release a lock when there is no contention. Unlike the others, this measurement is not instance specific and is currently measured once at the start of execution for each type of lock in the system.

Critical Region Time: For a specific instance of a lock, the average amount of time spent in the critical region, after the lock has been acquired, and before the lock has been released.

Waiting Time: For a specific instance of a lock, the average amount of time that each thread spends waiting to acquire it.

Read/Write Ratio: For a specific instance of a lock, the ratio of readers to writers is calculated.

From these three values, we can identify the context in which the lock is operating. A pattern language has been defined by others from which an appropriate synchronization primitive can be chosen. The following parameters define the context in which a lock operates:

Contention: The ratio of the time spent waiting to enter the critical section to the time spent actually executing in the critical section. A lock is considered highly contended if the ratio of wait to execution time is more than 10%–20%.

Granularity of parallelism: Granularity of parallelism is important because fine-grained parallel designs are very sensitive to locking-primitive overhead. Consider a critical section coarse-grained if the lock overhead is less than 10% to 20% of the average critical-section overhead.

Read-to-write ratio: Read-to-write ratio is the ratio of the number of read-only critical sections to the number of critical sections that modify data. Read-to-write ratios of unity or less are low, ratios of unity to n are moderate, and ratios greater than n are high, where n is the number of CPU's.

A second profiling element has been constructed as well, the `ProfilingList`. Like the `ProfilingLock`, the `ProfilingList` was developed to implement the interface of the element it profiles — in this case, the `List` class.

The `ProflingList` keeps track of how the list instance being profiled is being accessed. For instance, it counts the number of append operations, insertion operations, random access operations, etc.

5.2 Task 2: Modeling & development of control strategies

At the writing of the interim report, general aspects of control were the focus of this effort. As of the meeting with Dr. Janos Sztipanovits, our focused was narrowed to a specific area of interest. Therefore our view of control strategies has focused as well.

Initially, much of the work done for this task was research relating to general control strategies. Research systems have used various control schemes, including fuzzy logic, neural nets, and classical control theory.

After refocusing on specific aspects of parallel systems, control has been simplified into simple logic. Essentially, Elastic components use profiling to understand the access patterns that they are being subjected to. This data is then compared to known characteristics of the component set available and the best match can be selected and inserted.

Accomplishments:

- *Researched previous efforts*: We have looked at techniques used by other systems in order to understand the costs and benefits of as many control strategies as possible.
- *Implementation of various SMP locks*: Various implementations of SMP spin locks were constructed and dynamically profiled on several hardware and operating system platforms. Using the known characteristics of the choices available on each platform, a selection of the best lock could automatically be made.
- *Implementation of several Container classes*: Several implementations of a List interface were constructed and dynamically profiled. Implementations included an array based list and a doubly linked list. Performance characteristics of the implementations could be automatically analyzed on each platform, and each instance of the List could be profiled as well.

5.3 Task 3: Workbench for constructing self-adaptive software systems

As the previous sections in this report indicate, many pieces of the prototype Elastic Toolkit have been implemented. These pieces were implemented to explore the various aspects of feasibility required to be studied by an SBIR Phase I activity. Our conclusion is that building the Elastic Toolkit is feasible and will allow parallel applications to achieve high levels of performance without requiring tedious tuning activities.

Following are some of the key pieces of the Elastic Toolkit that have been constructed:

Locks: Locks are a critical piece of any SMP program. Sensitivity to locking is a common problem. Early experiments with locking showed that a synthetic application could gain a 13x speedup simply by changing the type of spin lock used. All of these locks implement the same interface and are therefore transparently interchangeable.

- `PthreadMutexLock`: This lock is based on the call `pthread_mutex_lock` provided by the `pthreads` library, common on many operating systems.
- `RWLock`: This lock implements a queued reader/writer lock based on the PthreadMutexLock. Multiple readers are permitted run in parallel, but must wait when a writer is accessing a resource. Likewise, a writer must wait for all readers to complete before altering the value of the resource.
- `SemaphoreLock`: This lock is based on the POSIX 1003.1b call `sem_wait`.

- `X86FIFOLock`: This lock is a queued lock based on x86 instructions; the intent is to provide fair[1] access to the critical region.
- `X86HardwareLock`: This lock is based on the atomic x86 instruction `xchgl`.
- `SparcHardwareLock`: This lock is based on the atomic Sparc V8 instruction `swap`.
- `LinuxOSLock`: This lock uses primitives defined by the Linux operating systems for simple spin locks. Its implementation varies from hardware platform to hardware platform, but is generally only a few assembly instructions.

Parallel-Safe List Structures: These are lists than can be accessed safely by multiple threads. All of these data structures implement the same interface and are therefore transparently interchangeable.

- `ArrayList`: An array based list. Implements thread safety either through entire list locking, or per element locking.
- `DynamicList`: A dynamically linked list. Implements thread safety either through entire list locking, or per element locking.

Timers: As previously discussed, timers are an important part of understanding the runtime behavior of a program. All of these timers implement the same interface and are automatically selected by the elastic System class at runtime for optimum performance.

- `GetTimeOfDay`: This timer uses the POSIX call `gettimeofday` for platform dependent accuracy timing. This is the least accurate of the timers and is used only as a fallback.
- `PentiumTSC`: This timer is based on an Intel Pentium assembly instruction that accesses a register on the Pentium that counts the number of clock cycles since bootup. This is a very accurate time measurement - as accurate as the microprocessor clock on a PC, with resolution in the nano-second range.
- `SolarisHRTime`: This timer is based on the Solaris Operating System call `gethrtime`. This OS call delivers resolution in the micro second range.

Pthread replacement library: A pthreads compatible library has been developed to allow the insertion of Elastic into applications written in "C" without modification. The application is simply linked with a C++ compiler, and the Elastic pthreads library is substituted for the system libpthreads. This has allowed insertion of Elastic into real applications in less than 15 minutes.

5.4 Task 4: Application Demonstration

Several applications were "Elasticized" and demonstrated. Results have been previously presented in a white paper that was distributed, in the interim report, as well as new results appearing in this document. Results have been extremely promising on all counts.

Accomplishments:

- Developed synthetic locking application: A synthetic application was developed where a number of threads contended for a single lock. Running this application on a dual processor Pentium Pro system, one could gain a speedup of 13X in runtime by selecting

[1]Fair in this context means first in, first out.

a different locking algorithm. Note that both of the locks compared were from system libraries and both are used in real applications. In addition, neither lock was consistently superior across OS kernel versions and libc versions.

- `tochnog`: `Tochnog` is a multi-threaded finite element analysis program. It is an interesting choice for several reasons — it is open source and the code is well organized, so insertion of Elastic technology was easy; in fact, it took less than 15 minutes. Also, `tochnog` comes with over 100 input files describing real problems. These files vary in analysis time from a few seconds to more than a day. Detailed results of the `tochnog` experiments can be found in the white paper "A Look at the Elastic Project".
- Xaos and mySQL: These are `pthreads` applications that were "Elasticized" using the Pthread replacement library. Both applications were converted to the Elastic Toolkit in a matter of minutes, by changing one include file, and linking in the replacement library.

5.5 Task 5: Measurement of effectiveness of self-adaptive systems

In this section, we report the results of preliminary testing of the prototype Elastic Toolkit. The results of this testing show that there is great potential in the area of self adaptive software construction.

Accomplishments: Experimental data has been collected for each of the following systems. The details of the experiments are in the following sections.

- Synthetic Results
- `Tochnog` results

5.5.1 Synthetic Benchmarks

A number of experiments have been performed using the Elastic Toolkit. These experiments are designed to create a controlled environment in which the various Elastic locking primitives may be evaluated. The data collected from this synthetic environment is used to form the ruleset used to select optimal locking primitives for real programs. The synthetic environment provides control over lock contention, granularity, and reader/writer ratios. As discussed in the white paper, these are three of the main forces that effect the performance of locking primitives.

An interesting result of profiling the locks with the synthetic environment occurs when examining the `PthreadMutexLock` and the `RWLock`. Both the `PthreadMutexLock` and the `RWLock` are based on the `pthread_mutex_lock` function provided in the `pthreads` library. The `RWLock` implements a reader/writer lock by maintaining a thread-safe queue that keeps track of desired accesses to the protected resource using a number of `pthreads` primitives.

Benchmarks performed in the synthetic environment showed that the `PthreadMutexLock` had relatively constant performance characteristics with respect the the three forces: granularity, contention, and reader/writer ratio. The `RWLock`, on the other hand, exhibited dynamic behavior that ranged from 100% speedup of the application to a 50% slowdown.

The `RWLock` has a free locking time of 1.5 uSec, while the `PthreadMutexLock` has a free locking time of 1.0 uSec. This would indicate that under many circumstances, the performance of the `RWLock` should be far below that of the `PthreadMutexLock`. However, the `RWLock` will allow multiple readers to access a resource in parallel, thus completely eliminating lock overhead when this situation occurs. To further complicate this relationship, the amount of contention placed upon a `RWLock` effects its performance, because during periods of low contention, each reader and writer

experience the full overhead of the lock, and do not derive any benefit from the parallel nature of the lock.

The following table depicts the ruleset for choosing the `RWLock` over the `PthreadMutexLock`. A "+" indicates that the `RWLock` performs better than the `PthreadMutexLock` in the same context, while a "-" indicates that the `RWLock` has worse performance. This table was derived from a trial on a 4-way Solaris machine.

	Lock Contention		
R/W Ratio	High	Medium	Low
High	+++	+	-
Medium	++	-	- -
Low	- - -	- -	- - -

This experiment demonstrates that the use of the Elastic Toolkit can provide significant performance enhancements for applications with dynamic locking contexts. The abstract interface for Elastic locking primitives allows the programmer to specify "lock for read" and "lock for write" when a resource is acquired. By monitoring the read/write ratio, and the contention placed on a resouce, the Elastic Toolkit can select the locking primitive with the optimal performance for the current context.

5.6 `tochnog`

One of the experiments with `tochnog` included the insertion of the various types of locks supplied by Elastic into the single mutex that `tochnog` defines. A summary of some of the data follows:

Test Case	unmodified	x86HardwareLock	x86HardwareLock w/ BackOff	SemaphoreLock
wave1	11.303s	10.792s	11.070s	16.786s
examp13	18.382s	18.245s	20.549s	24.081s
examp14	4m42.874s	4m34.229s	4m37.834s	5m26.970s

All tests were performed on a dual processor Pentium Pro machine, running Linux 2.2.10, with two active threads and low background activity.

wave1: calculates a one dimensional wave function.

examp13: simulates a tendon embedded over a circular solid, with an initial stress on the tendon that will deform the solid. The calculation finds the final deformation of the solid.

examp14: simulates a metal sheet being subject to an extrusion process. The calculation estimates the final thickness of the metal sheet and the build up of plastic strains within the sheet.

This data shows that for a high contention lock, simple replacement with another type of spin lock does not have a significant affect on performance.

This is the expected result. Replacing the original `pthreads_mutex` calls with an Elastic `X86HardwareLock` boosts the performance by only a small amount, due solely to the efficiency of the implementation.

On the other hand, the Elastic `SemaphoreLock` is a poor performer in this context. This lock is based on a POSIX system call and it would be a logical choice for this type of lock; luckily in this case the author who wrote `tochnog` did not make such a choice.

5.7 Task 6: Document the Phase I Effort

Documentation of the Phase I effort has occurred in several main areas. One area is the research conducted at the beginning of the project. Many papers have been collected, notes taken, etc. in conducting the research. This body of materials documents past efforts made on many topics relevant to Elastic.

A second area includes the reports written for the project, as well as the white paper which was written and distributed to key individuals. These works include specific information about the project as well as status and results.

The last area includes design documentation generated in designing the prototype Elastic architecture, mainly in the form of C++ source code, and accompanying documentation. This documentation will form an API specification and guide for the users of the Elastic Toolkit.

Accomplishments:

- Developed body of material in related areas.
- Design documentation for prototype system created.
- Submitted status report, final report, and white paper.

6 Summary and Plans

Clifton Labs has evaluated the goals, feasibility, and commercial viability of the Elastic project in the Phase I activity. As a result of these investigations, Clifton Labs finds that Elastic is both technically feasible and commercially viable. Much of our early effort in assessing feasibility went into researching past projects applicable to the problem domain. Researching other's efforts was part of our strategy in order to assess feasibility, market, competition, and other important aspects of product development. Commercial viability was established by using, in part, an informal survey to establish interest and willingness to purchase should such a product become available. The results from this survey were overwhelmingly positive and additional investigations discovered that other companies are already successfully marketing programming libraries. Thus, Clifton Labs believes that there is strong evidence supporting the technical and commercial feasibility of Elastic.

Clifton Labs is now planning a full scale development of an Elastic library. Early prototyping has been completed and evaluated. In planning the full development, Clifton plans to:

- Seek continuation of the SBIR funding to phase II.
- Expand and further optimize the Elastic library.
- Identify and secure participation of commercial partners for alpha testing.
- Identify and secure participation of commercial partners for beta testing.
- Identify and evaluate strategies for distribution and sale of the Elastic library. Possible partnering arrangements for marketing and sales may be desirable.
- Plan for phase III activities.